Building the Basic Structure of a Windows Help File

BY RAY DUNCAN

The authoring of a Windows help file is an extremely important part of any Windows application development project. Unfortunately, it is the part of the project that often gets slighted or postponed until very late in the development cycle. Additionally, the design and implementation of a high-quality help file is not a task the average programmer is well qualified to pursue, regardless of his or her other talents. The Windows Help System can be a powerful ally but, when help files are put together by amateurs, the results can be truly grotesque.

In the best case, help file design and development should be carried out by a team that includes the application designer, a technical writer, an experienced manuscript editor, a graphics designer/typographer, and a user-interface designer. Of course, only big companies like Microsoft have the luxury of deploying such a formidable team of experts against a help file project. The rest of us must simply do the best we can, using help files created by experts as models and seeking advice from more experienced developers where we can find it. Again, I strongly recommend that you obtain the Microsoft Developer Network CD-ROM and carefully study the "Help File Authoring Guide" included on that disk.

In the last issue, we discussed some general philosophical considerations regarding help files, as well as components of the help file system. In this column, we'll move forward with the implementation of a help file for the EXELOOK program that was published earlier this year in several Power Programming columns.

DESIGNING THE HELP FILE Probably the most important step in help file development is the one you carry out with a pencil and paper before you even touch a computer: the design of your help file's logical structure. (The logical structure is the structure that the end user perceives; not the actual, physical structure.) Several objectives need to be kept in mind during this process.

- Help screens should be clean, attractive, and uncluttered (avoiding excessive use of different fonts, small fonts, and lots of colors).
- The organization of the help file should reflect the user's expectations as well as the organization of the program it describes.
- The user must be able to navigate through the help file quickly and easily; there should be no surprises or dead ends.
- The user should be able to move quickly from task-oriented overviews to detailed information about a specific feature and vice versa.
- There should be multiple routes to any desired chunk of information.
- Every topic in the file should be searchable under all possible synonyms and aliases.
- No feature of the program should be left unmentioned.

This practical example of the construction of a help file outlines the basic structure of help files and the steps to follow to implement them in your own applications.

To start, you might just write down a list of all the subjects and aspects of your program that deserve an individual explanation in the help file. For example, there should be an overview of what the program does, an entry for each of the program's menu items, and a description of each task that can be accomplished with the program. Once you've completed such a list, try to gather its components into a relatively small number of groups—a group of menu items, a group of tasks, and so on.

The second step is to sketch out the relationship between the help file's table of contents and the various chunks of information in the help file. Since one of the objectives is to keep screens uncluttered, you want to avoid table of contents screens that have more than, say, eight or ten distinct items. Thus, in most cases, you'll end up with a multilevel table of contents where the initial contents screen for the help file will lead to a more specialized table of contents for each of the subject groups, and thence to the actual help text. My own sketch for the organization of the help file for the EXELOOK utility is shown in Figure 1. As you can see from the figure, you can think of the help file's logical organization at this early stage as being much like a decision tree (but certainly not a binary tree or a balanced tree).

CREATING A HELP FILE SKELETON Now you're ready to lay hands on the keyboard. Create a new directory that will hold all the files associated with your help file. (Doing this will allow you to back up the files as a set easily before you make any major changes.) Fire up your word processor and create an empty document file for the table of contents, and one file each for the branches of the tree under the table of contents. In the case of Figure 1, you'd create four files that

you might name CONTENTS.DOC, OVERVIEW.DOC, MENUS.DOC, and TECHINFO.DOC.

Once all the empty files are created, it's easy to turn them into a skeleton of the final help file. Within each file, type a series of lines that will ultimately be the headings for all chunks of information held in that file. The CONTENTS.DOC file is no different from the others in this respect, since the contents screens have no special status from the point of view of the help file viewer (except that one of them is designated as the first screen to be displayed when the help file is opened). For example, you might enter the following lines into the CONTENTS.DOC file.

```
Table of Contents
Index to General Information
Index to Menu Descriptions
Index to Technical Information
```

And for MENU.DOC you might enter these lines:

```
File Menu
Display Menu
Help Menu
```

Next, insert a hard page break before the first line, between each line, and after the last line of each file. These hard page breaks tell the help file compiler where to divide one help file chunk of information from another. The text between two page breaks is presented as a single unit, called a topic, within the help file viewer's window. The first line of a topic, which is the topic's heading as far as the end-user is concerned, has no special status for the help file compiler or help file viewer program; you're just using topic headings for now as placeholders, and as something recognizable for you to look at when you're compiling the help file skeleton.

Now it's time to assign "context strings" to each topic, including the topics in CONTENTS.DOC. Context strings are merely labels that the help compiler can recognize and resolve as the target of a hyperlink jump from elsewhere in the file. A context string is associated with a topic—that is, the topic is assigned a symbolic label for the help compiler's use—by creating a footnote with the special footnote character # at the very beginning of the topic. If you're using WinWord, you must position the caret immediately after the page break that delimits the beginning of the topic, select Insert Footnote, type the # character as the footnote character in the dialog box, then type the desired context string as the contents of the footnote. Keep in mind that the context string cannot contain embedded blanks.

Help File Structure

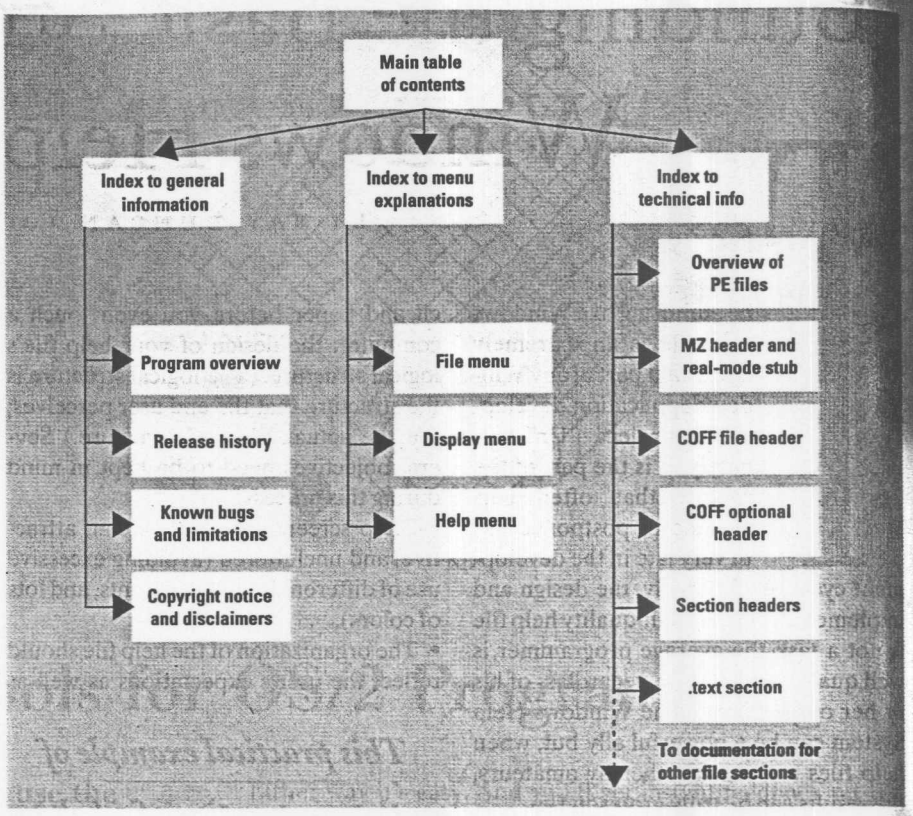


Figure 1: **This depicts the organization of the help file for the EXELOOK.EXE utility.**

You want to pick context strings that are unique and easy to remember, and assign them according to some simple convention. For example, the context strings for the topics located in CONTENTS.DOC might be assigned as follows:

```
contents_main
contents_overview
contents_menu
contents_techinfo
```

while those for topics in MENU.DOC might be assigned like so:

```
menu_file
menu_display
menu_help
```

At this point, you've got a set of very odd-looking document files where each file consists of pages that are empty except for a single line at the top, with each page's single line preceded by a # footnote indicator, and with a single footnote located at the bottom of each page. If you're editing MENU.DOC in WinWord 2.0 with the footnote window open, you'll see something like the screen shot shown in Figure 2.

Now go back to CONTENTS.DOC. Amplify the main and subsidiary tables of contents so that the first page of the file contains the following text.

```
Table of Contents

Index to General Information
Index to Menu Descriptions
Index to Technical Information
```

The second page should look like this:

```
Index to General Information

Program Overview
Release History
Known Bugs and Limitations
Copyright Notice
```

The third page of CONTENTS.DOC should say

```
Index to Menu Descriptions

File Menu
Display Menu
Help Menu
```

And I'll leave the text of the fourth page to your own imagination.

INSTALLING THE FIRST HYPERLINKS You are now ready to add the hyperlinks in CONTENTS.DOC that will let you jump down through the levels of the table of contents and from there to the leaves of the help file topic tree. A simple text hyperlink is created by formatting the text that will be visible to the user as the hyperlink with a double-underline (by default, the hyperlink text is displayed in green with a single underline), then inserting the context string that is the destination of the hyperlink as "hidden text" immediately after the hyperlink text with no intervening spaces. The contents of the text that is visible to the user as the hyperlink can be anything, but the context string that is inserted as hidden text must exactly match the context string that was assigned to the destination topic with the # footnote. The first page of the CONTENTS.DOC file, after inserting hyperlinks, should look like Figure 3 if you're using WinWord.

When everything seems satisfactory in each of the four document files, save the files in the their native word processor format and additionally export each of them as a Rich Text Format (RTF) file. You should now have eight files in your help project directory:

```
CONTENTS.DOC
CONTENTS.RTF
OVERVIEW.DOC
OVERVIEW.RTF
MENU.DOC
MENU.RTF
TECHINFO.DOC
TECHINFO.RTF
```

Throughout the remainder of the help file development cycle, you'll only edit the document (.DOC) files. The RTF files are the input to the help file compiler so you'll need to re-export them each time you rebuild the help file for testing, but they have no other purpose.

COMPILING THE HELP FILE SKELETON You're almost ready to compile the help file skeleton. But before you can run the help file compiler (HC31.EXE), you have to create a help project (HPJ) file that tells the compiler which files to process, the topic name for the main "contents" screen, the title for the help file window, and other crucial information. The possible directives that can be placed in an HPJ file were summarized in the last column; for our purposes here you can create a simple help project file named EXELOOK.HPJ that reads like this:

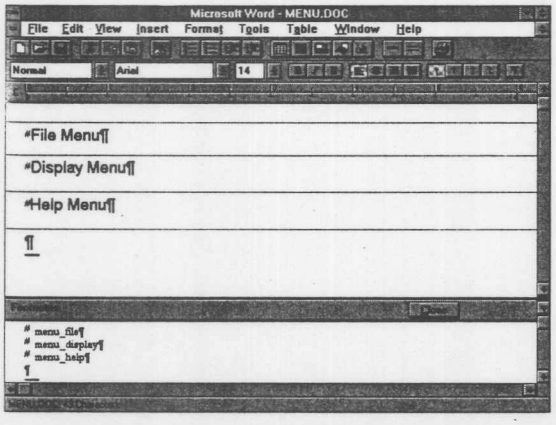
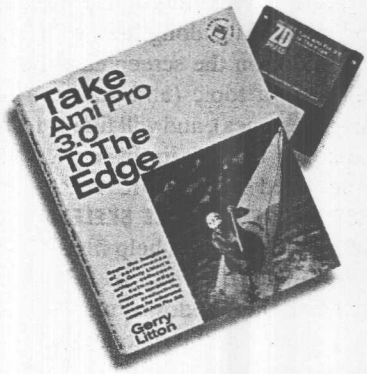


Figure 2: **This screen shot shows Word for Windows 2.0 while editing the MENU.DOC file skeleton, with topic separators (hard page breaks), topic headings, and topic labels (context string footnotes) in place.**

```
[OPTIONS]
CONTENTS=contents_main
TITLE=EXELOOK Help
[FILES]
CONTENTS.RTF
OVERVIEW.RTF
MENU.RTF
TECHINFO.RTF
```

The HPJ file must be saved as a plain ASCII text file; it should not be saved as a word processor document (.DOC) file or a rich text format (RTF)

The Cutting Edge



ISBN: 1-56276-089-0
Price: $29.95

Attention Ami Pro users! Maximize your productivity with this in-depth book filled with insider tips and shortcuts. Gerry Litton shows you how to become a power user by taking advantage of Ami Pro's advanced tools, including power fields, macros, and automated style sheets.

Available at fine bookstores, or call 1-800-688-0448 ext. 8452

ZIFF-DAVIS ZD PRESS

© 1993 Ziff-Davis Press

file. Make sure that the HPJ file is in the same directory with the RTF files, then run the help file compiler with the name of the HPJ file as the command line parameter. For example:

```
C>HC31 EXELOOK <Enter>
```

Note that the extension HPJ is assumed and need not be entered. The help file compiler will chug along for a while, displaying a dot on the screen each time it crunches up a topic (and maybe a few warning messages), and will finally terminate, leaving behind, in this case, a new help file called EXELOOK.HLP.

TESTING THE HELP FILE SKELETON You can test your compiled help file skeleton EXELOOK.HLP with the File-Run command in the Windows Program Manager. Enter either the path and name of the help file alone, or the name of the Windows help file viewer (WINHELP .EXE) followed by the name of the help file as the command line parameter. If all goes as expected, the help file viewer should appear with the first page from CONTENTS.DOC visible in the main window (this was controlled by the CONTENTS= directive in the HPJ file) and the text specified by the HPJ file's TITLE= directive in the window's title bar. (See Figure 4.)

You should now navigate up and down each of the branches of the help file tree, using the hyperlinks displayed in green and the "Back" button under the WINHELP.EXE menu bar, until you are satisfied that all the fundamental linkages and sections of the help file's logical tree are present and working as expected. Try to put yourself in the end-user's shoes, and make sure that the hyperlinks describe their destinations clearly and completely.

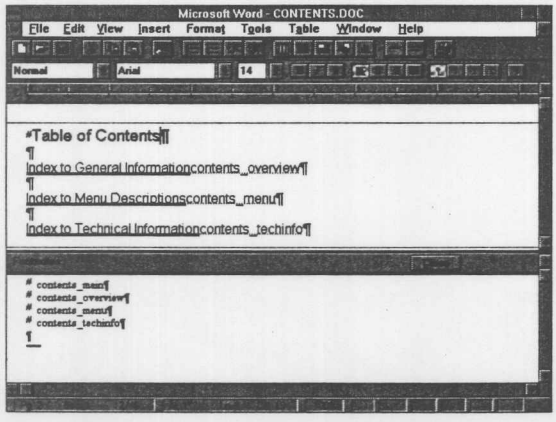


Figure 3: **This is a screen shot of Word for Windows 2.0 while editing the CONTENTS.DOC file, showing the first page of the file with hyperlinks added.**

FILLING IN THE SKELETON Close the help engine now and return to your word processor. Open up each of the help information source files—OVERVIEW.DOC, MENU.DOC, and TECHINFO.DOC—and flesh out each topic by adding explanatory text under the topic's title. Write clearly but informally, avoid lengthy paragraphs, run-on sentences, and technojargon, and keep each paragraph about the same length. Don't get carried away with fonts or colors; a sans serif font such as Arial in 12- to 18-point bold is good for topic titles, and 10- or 12-point plain Arial is fine for the body text. Other fonts should be reserved for extraordinary occasions, and colors other than black should almost always be avoided.

Don't worry about the margins when you compose the topic's body text—just set the margins so you can see what you are writing. The help file viewer mostly ignores your margins anyway (except for left-hand indents); if the text is too wide to fit into the help viewer's window, each paragraph is "reflowed" so both margins are visible (with a couple of exceptions that we'll discuss in the next issue). Bring up a help file in the help file viewer and play around with resizing the viewer's window to see this effect for yourself.

If the text of a topic is too long to fit into the help viewer's window, a scroll bar is added to the window and the user can move through the topic by clicking the scroll bar or dragging the thumb. The body of a topic should be neither too short nor too long, and this is a judgement that becomes easier with experience. Topics that consist of only one or two paragraphs are generally too short. (These can be better implemented as pop-up windows, which we'll also discuss in the next column.) Topics that are more than twice the length of an "average-size" help viewer window are probably too long and should be subdivided.

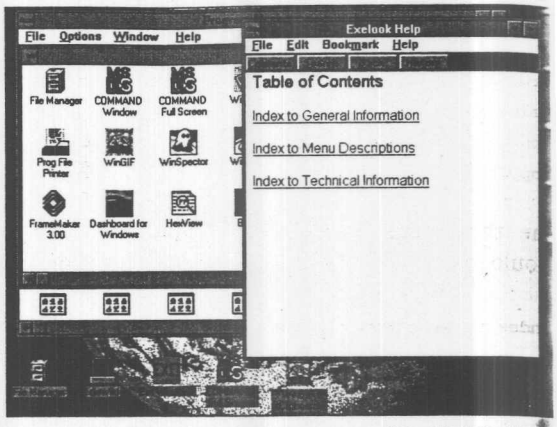


Figure 4: **This shows the main contents page of the EXELOOP.HLP help file skeleton, viewed with WINHELP.EXE.**

When you're done editing the first version of the complete help text into the document files, save the files in their word processor document format. Then export them again as RTF files and rerun the help file compiler. View the results of your labors (the recompiled EXELOOK.HLP file) with WINHELP.EXE. Now you'll become acutely aware of the first hard lesson of help file development: The look of the help text within the word processor—particularly in terms of spacing and alignment—is not a good predictor of how that same text will appear in the Windows help viewer. The second hard lesson is that getting the text to look the way you want takes seemingly endless twiddling. And the third hard lesson is that the help file compiler is agonizingly slow.

In the next issue, I'll show you how to add code to your application to activate its help file, then we'll move on to more sophisticated help file facilities such as bitmaps, keywords for searches, pop-up windows, secondary windows, and macros. Meanwhile, you can download my versions of the DOC, RTF, and HPJ files that form the basis of EXELOOK .HLP from the Programming Forum on PC MagNet, archived as EXEHLP.ZIP.

THE IN BOX Please send your comments, suggestions, or questions to me at any of the following e-mail addresses:
PC MagNet: 72241,52
MCI Mail: rduncan
Internet: duncan@cerf.net □